Java IO Streams
Part 1

By Fadel K.
www.fadelk.com

File
FilenameFilter

IO Operations in Standard IO package (java.io)

- Programs read inputs from data sources (e.g., keyboard, file, network, or another program) and write outputs to data sinks (e.g., console, file, network, or another program).
- Inputs and outputs in Java are handled by the so-called stream. A stream is a sequential and continuous one-way flow of information (just like water or oil flows through the pipe).
- It is important to mention that Java does not differentiate between the various types of data sources or sinks (e.g., file or network). They are all treated as a sequential flow of data.

IO Operations in Standard IO package (java.io)

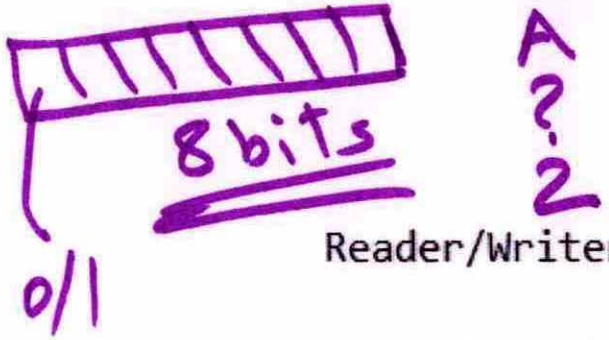
- The input and output streams can be established from/to any data source/sink, such as files, network, keyboard/console or another program.
- The Java program receives data from data source by opening an input stream, and sends data to data sink by opening an output stream.
- All Java I/O streams are one-way (except the RandomAccessFile, which will be covered later). If your program needs to perform both input and output, you have to open two separate streams - an input stream and an output stream.

IO Operations 3 Major Steps

- **IO operations involved three steps:**
  1. **Open an input/output stream associated with a physical device (e.g., file, network, console/keyboard), by constructing an appropriate IO-stream object.**
  2. **Read from the opened input stream until "end-of-stream" encountered, or write to the opened output stream (optionally flush the buffered output).**
  3. **Close the input/output stream.**

IO Universes

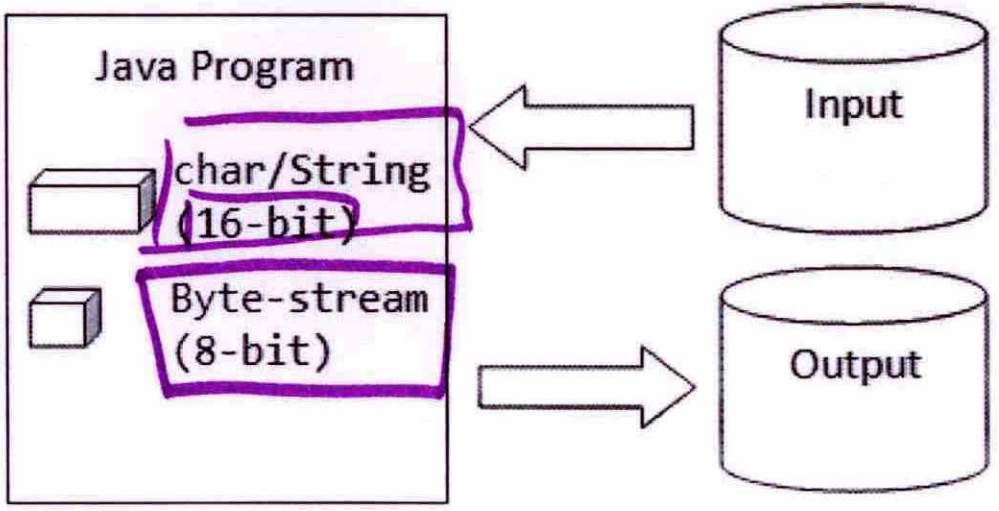
- **Java IO works in two different universes**
  - **Character based IO: for16-bit (2 bytes) streams, required when dealing with character sets (i.e. Text).**
  - **Byte based IO: for 8-bit (1 byte) streams, required in fact for everything ☺**



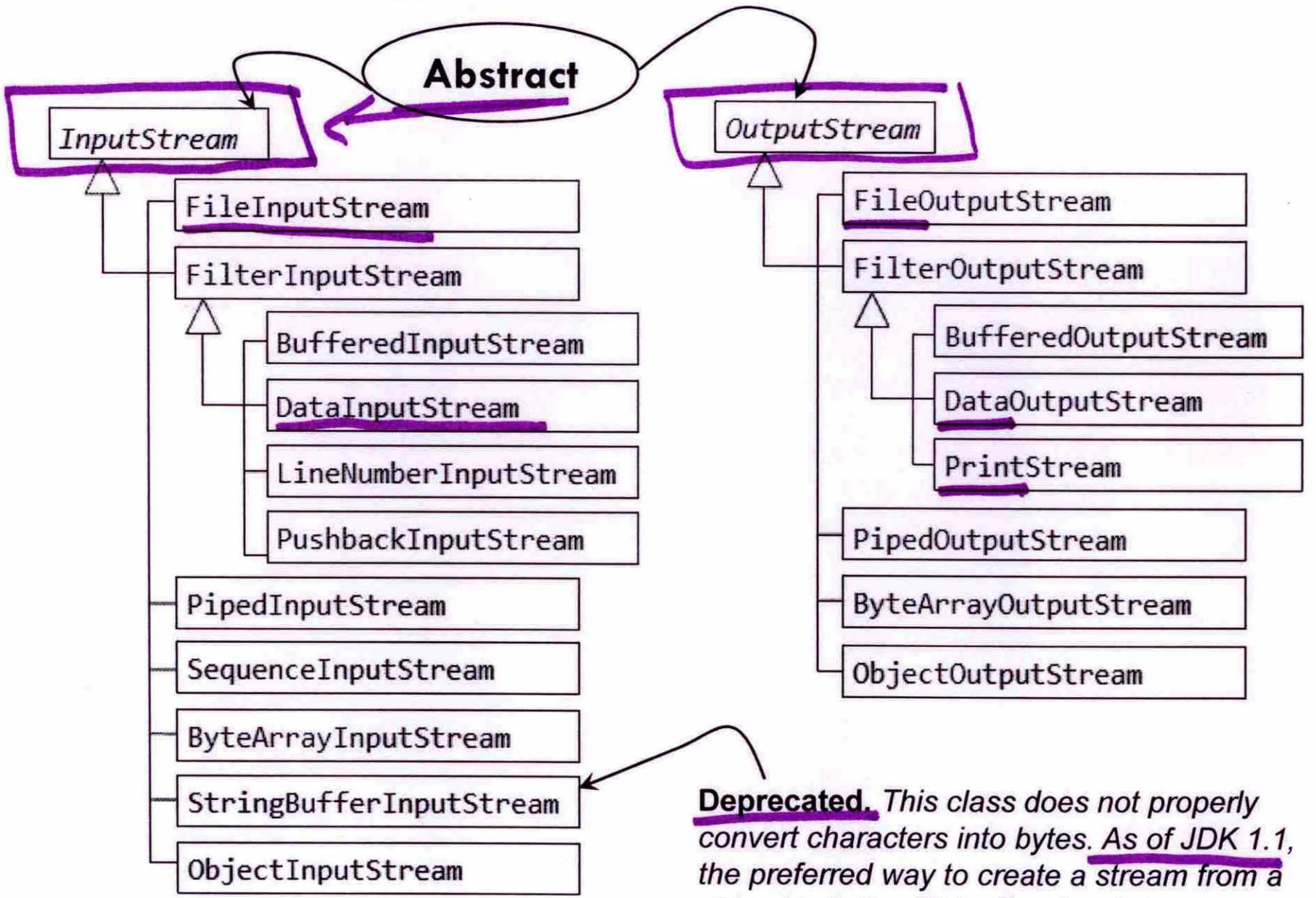
Reader/Writer

InputStream/OutputStream

- Binary Files
- Text Files (ASCII, UCS-2/UTF-8) (System-dependent)



Byte-Based IO & Byte Streams



Input & Output Streams

- Byte streams can be used to read or write bytes serially from an external device.
- All the byte streams are derived from the abstract superclass InputStream and OutputStream, as illustrated in the previous class diagram.
- The abstract superclass InputStream declares an abstract method read() to read one data-byte from the input source, and convert the unsigned byte value (of 0 to 255) to an int.

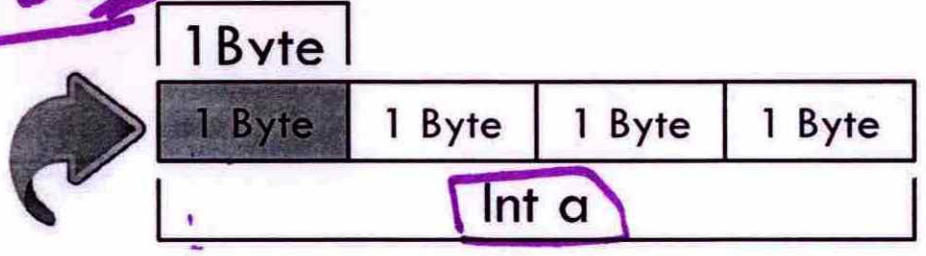
int = 4 bytes

InputStream Read() Method

- The read() method will continue reading:
  - Until a byte is available,
  - An I/O error occurs,
  - Or the "end of stream" is reached.
- It returns -1 if for "end of stream", and throws an IOException if it encounters an I/O error.

```
// read one data-byte from the input stream
public abstract int read() throws IOException
```

0 1 2 3 ... 255

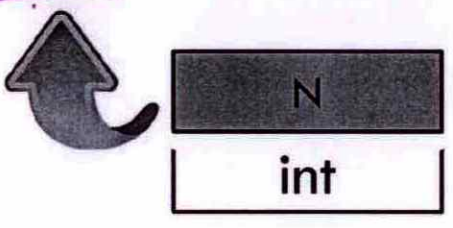


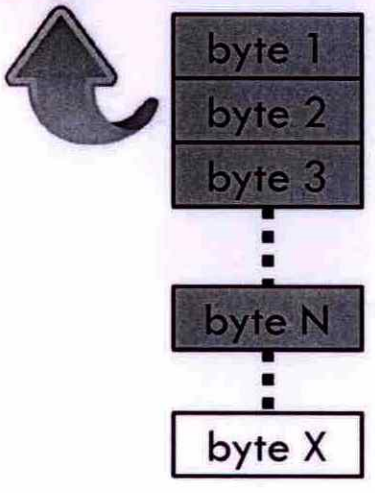
InputStream Read(byte[])

- **First variation of read() method is implemented in the InputStream for reading a block of bytes into a byte-array buffer. It returns the number of bytes read, or -1 if "end-of-stream" encounters.**

```
// Read from a source into an array of bytes

public int read(byte[] bytesBuffer) throws IOException
```

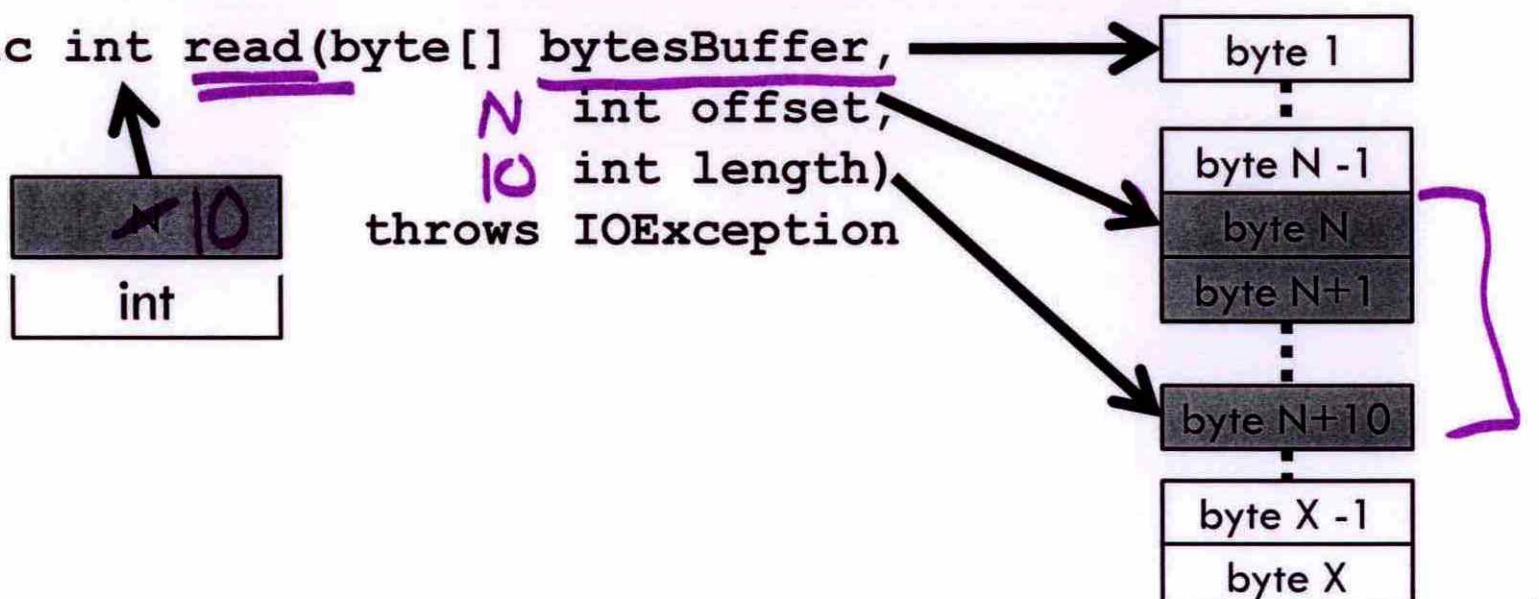




InputStream Read(byte[], int, int)

- **Second variation of read() method is implemented in the InputStream for reading a block of bytes into a byte-array buffer, to fill the array partially. It returns the number of bytes read, or -1 if "end-of-stream" encounters.**

```
// Read "length" number of bytes,
// store in bytesBuffer array starting from index offset.
public int read(byte[] bytesBuffer,
                int offset,
                int length)
throws IOException
```

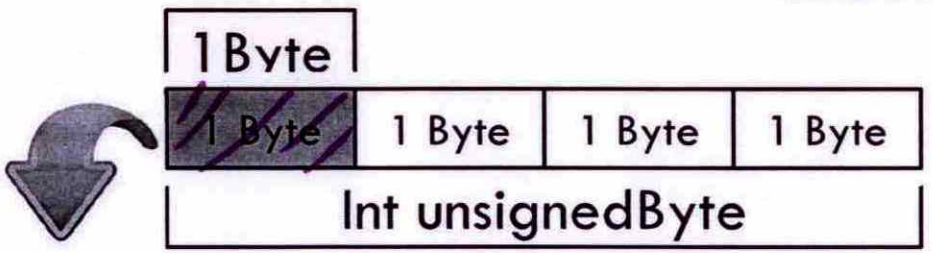


OutputStream Write() Method

- **Similar to the input counterpart, the abstract superclass OutputStream declares an abstract method write() to write a data-byte to the output sink. The least significant byte of the int argument is written out; while the upper 3 bytes are discarded.**

```
// write one data-byte to the output stream

public void abstract void write(int unsignedByte)
                                throws IOException
```



OutputStream Write() Method Cont.

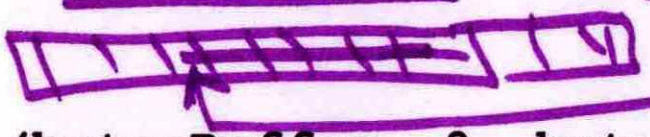
- **Similar to the read(), two variations of the write() method to write a block of bytes from a byte-array buffer are implemented**

```
// Write "length" number of bytes,
// from the bytesBuffer array starting from index offset.

public void write(byte[] Buffer, int offset, int length)
                                              throws IOException

// Same as write(bytesBuffer, 0, bytesBuffer.length)

public void write(byte[] Buffer) throws IOException
```



Input & Output Streams Closing

- Both the InputStream and the OutputStream provides a close() method to close the stream, which performs the necessary clean-up operations as well as frees the system resources. (Although it is not absolutely necessary to close the IO streams explicitly, as the garbage collector will do the job, it is probably a good practice to do it to free up the system resources immediately when the streams are no longer needed.)

```
// close the opened Stream
public void close() throws IOException
```

Moreover

- In addition, the OutputStream provides a flush() method to flush the remaining bytes from the output buffer.

```
// Flush the output
public void flush() throws IOException
```

- InputStream and OutputStream are abstract classes that cannot be instantiated. You need to choose an appropriate concrete implementation subclass to establish a connection to a physical device.

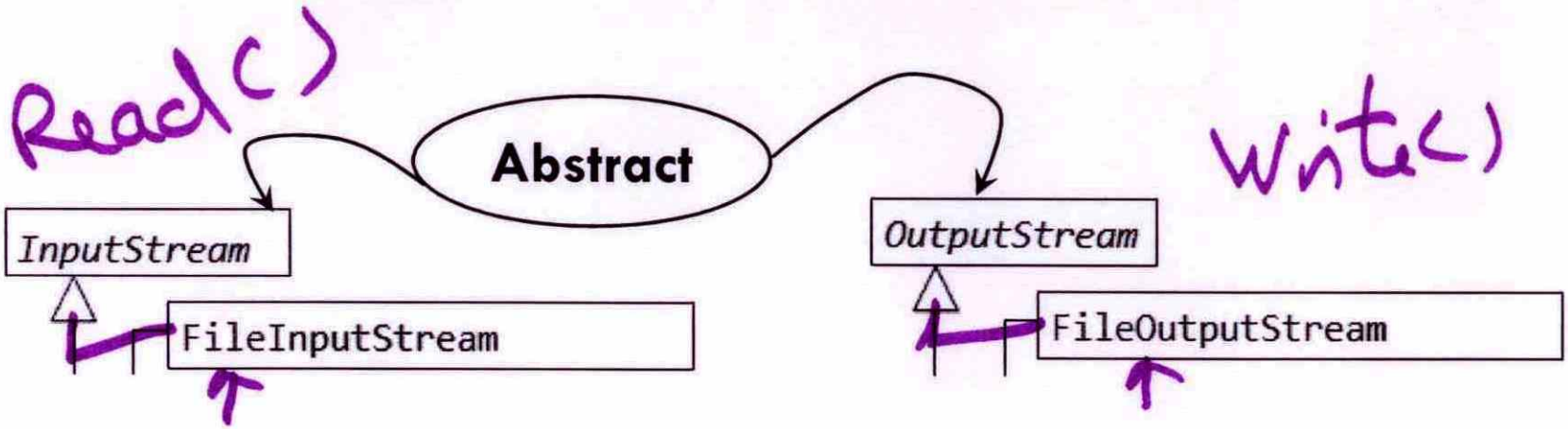

```
public class T1 {
 public static void main(String[] args){
  int a = 3;
  System.out.println( a > 4 );
  System.out.println( a = 5 );
  System.out.println( (a = a + 6) > 10);
 }
}
```

Output

```
false
5
true
```

FileInputStream & FileOutputStream

- FileInputStream and FileOutputStream are concrete implementation to the abstract class of InputStream and OutputStream, to support File IO.



java.io

Class FileInputStream

java.lang.Object
└ java.io.InputStream
└ **java.io.FileInputStream**

Constructor Summary

FileInputStream(File file) Creates a FileInputStream by opening a connection to an actual file, the file named by the File object file in the file system.
FileInputStream(FileDescriptor fdObj) Creates a FileInputStream by using the file descriptor fdObj, which represents an existing connection to an actual file in the file system.
FileInputStream(String name) Creates a FileInputStream by opening a connection to an actual file, the file named by the path name name in the file system.

java.io

Class FileOutputStream

java.lang.Object
└ java.io.OutputStream
└ **java.io.FileOutputStream**

Constructor Summary

FileOutputStream(File file) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(File file, boolean append) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(FileDescriptor fdObj) Creates an output file stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system.
FileOutputStream(String name) Creates an output file stream to write to the file with the specified name.
FileOutputStream(String name, boolean append) Creates an output file stream to write to the file with the specified name.

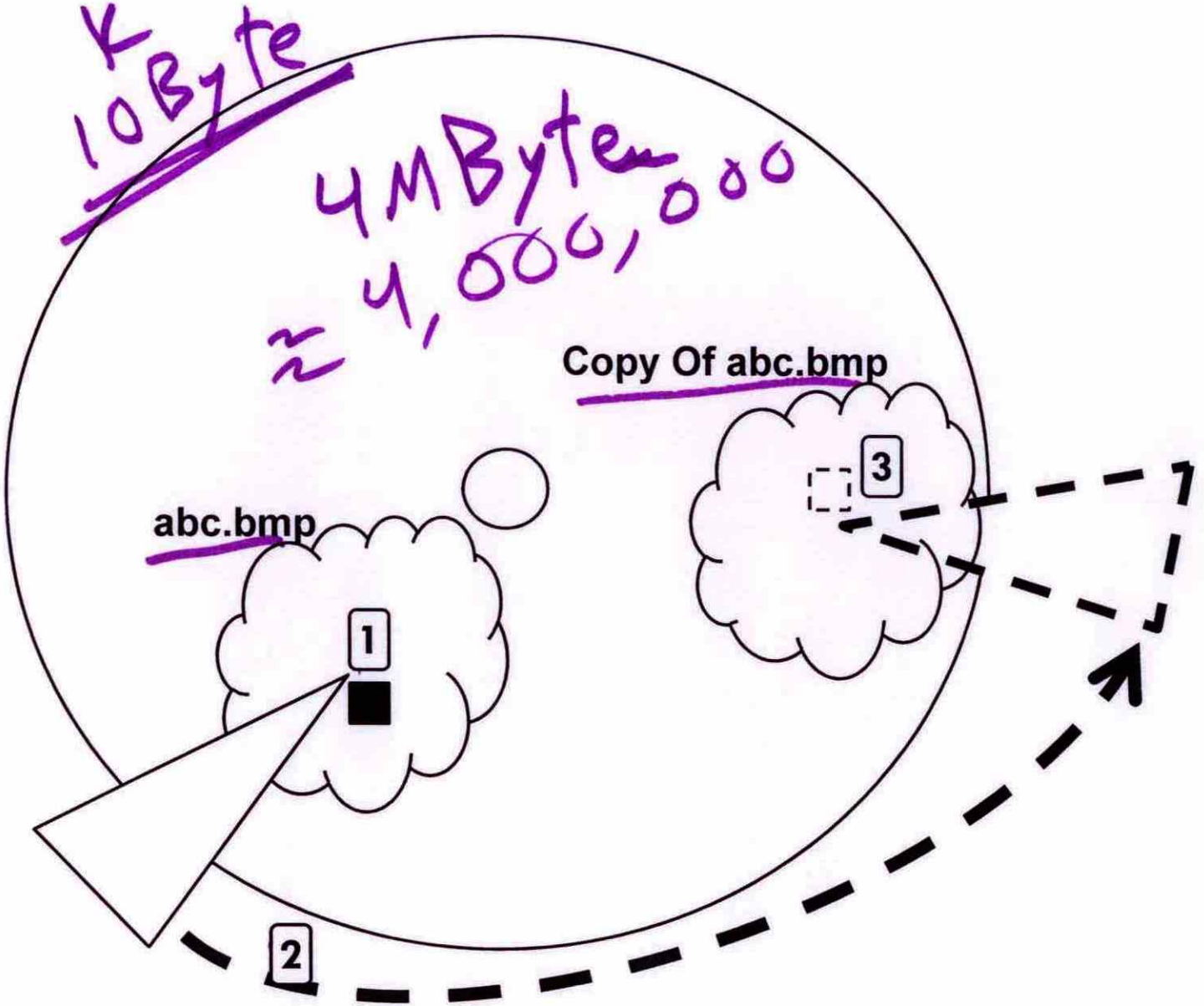
Example A

```
import java.io.*;
public class test {
 public static void main(String[] args) {
  try {
   String p="E:\\test\\"; // p for path
   String n="abc.bmp"; // n for name
   File fileIn = new File(p + n);
   FileInputStream fis = new FileInputStream(fileIn);
   FileOutputStream fos = new FileOutputStream(p+"Copy Of"+n);

   int a; // An integer of 4 bytes to hold a single byte.
   // Read a single byte into int a , write it back in fos.
   while (( a = fis.read() ) != -1) {fos.write(a);}

   fis.close();
   fos.close();
  } catch (Exception ex) {ex.printStackTrace();}
 }
}
```

E:\\test\\abc.bmp
File object
0-255
End of stream



4 : Repeat Steps 1,2,3 for each byte in the file